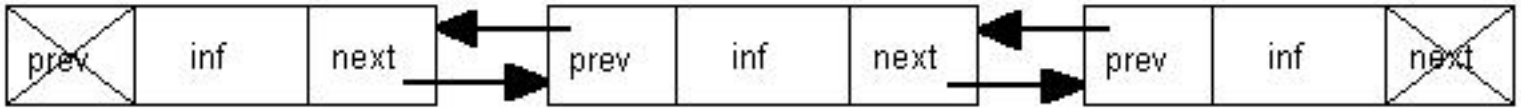
# Научно-исследовательский практикум

КЛАССЫ ВНУТРИ КЛАССОВ. ПЕРЕГРУЗКА ОПЕРАТОРОВ. НАСЛЕДОВАНИЕ.

# Немного теории – связный список

Связный список


Двусвязный список



# Можно ли поместить класс в класс?

- Что если нам нужен «нестандартный» тип данных?
- Например, мы хотим реализовать класс двумерных векторов – или точек в двумерном пространстве
- Самый распространённый вариант, как это сделать – написать свой класс
  - Конечно, такие классы уже реализованы в куче стандартных библиотек

```
struct Point2D
{
	Point2D () : m_x(0), m_y(0){}
	Point2D (double x, double y)
	{
		m_x = x;
		m_y = y;
	}
	double m_x, m_y;
};
```

# А как такой класс поместить в наш контейнер?

- Очевидно, что раньше мы хранили элементы double
- Теперь нам нужно хранить элементы Point2D
- Для этого мы во всём проекте изменяем double на Point2D
- Я приведу только изменения в заголовочном файле – реализация в .cpp – на ваше усмотрение

# Заголовочный файл с SimpleContainer

```
class SimpleContainer
{
public:
      SimpleContainer();
      SimpleContainer(int size, Point2D value=Point2D(0,0));
      SimpleContainer(const SimpleContainer& container);
      ~SimpleContainer();

      SimpleContainer& operator=(const SimpleContainer&);
      SimpleContainer operator+(const SimpleContainer&) const;

      int size() const {return m_size;}
```

# Заголовочный файл с SimpleContainer

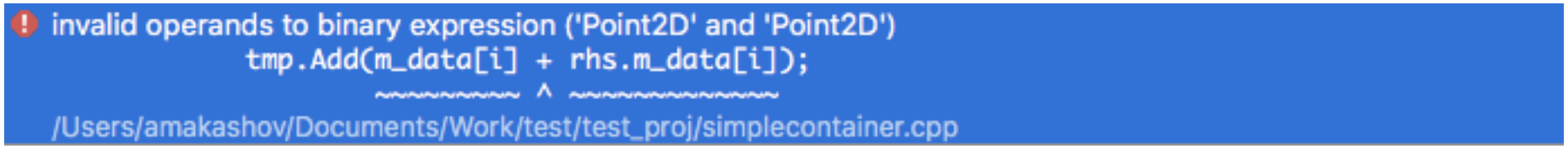
```
        int capacity() const {return m_capacity;}
        Point2D get(int index) const;
        void set(int index, double value);
        void Add(double value);

private:
        void Reallocate();

        Point2D* m_data;
        int m_size; // логический размер
        int m_capacity; // фактический размер
};
```

# Что получилось?

```
SimpleContainer SimpleContainer::operator+ (const SimpleContainer& rhs) const
{
        SimpleContainer tmp;
        if (m_size == rhs.m_size)
        {
                for (int i = 0; i< m_size; i++)
                        tmp.Add(m_data[i] + rhs.m_data[i]);
        }
        return tmp;
}
```



# И снова перегрузка операторов

- Нам не хватает возможности складывать объекты Point2D
- Нет – значит, нам придётся реализовать её самим
- Для этого – нужен operator+  для нашего класса Point2D
- Что наш оператор должен делать:
  - Он не должен изменять саму «исходную точку», то есть *this – то, что стоит слева от оператора +
  - Он не должен изменять то, что стоит справа от оператора, то есть rhs
  - Результатом должна стать новая точка

# Как изменился наш класс точек

```
struct Point2D
{
	Point2D () : m_x(0), m_y(0){}
	Point2D (double x, double y)
	{
		m_x = x;
		m_y = y;
	}
	Point2D(const Point2D& point) = default;
	Point2D operator+(const Point2D& rhs) const
	{
		...
	}
	double m_x, m_y;
};
```

# Функция main()

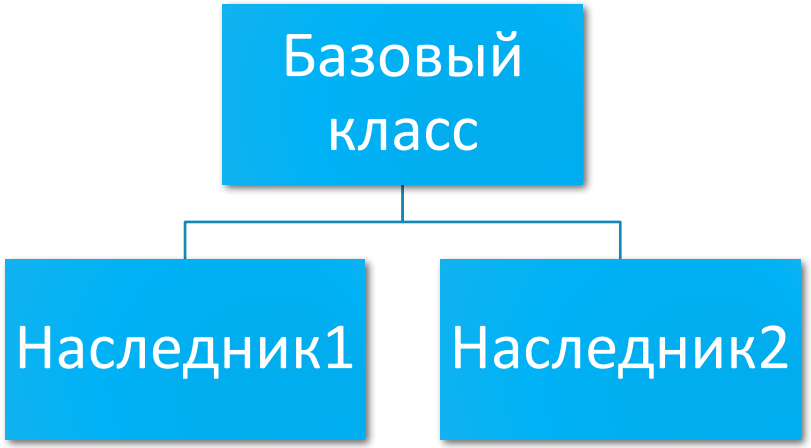
```
int main(int argc, char *argv[])
{
	SimpleContainer testContainer;
	for (int i=0; i<5; i++)
		testContainer.Add(Point2D(i,3-i));
	SimpleContainer test2 = testContainer;
	SimpleContainer test3 = testContainer + test2;
	for (int i=0; i<test3.size(); i++)
		cout << "X="<<test3.get(i).m_x << "\tY="<<
test3.get(i).m_y<< "\n";
}
```

# Вывод программы

```
Starting /Users/amakashov/Documents/Work/test/build-test_proj-Desktop-Debug/test_proj...
X=0         Y=6
X=2         Y=4
X=4         Y=2
X=6         Y=0
X=8         Y=-2
/Users/amakashov/Documents/Work/test/build-test_proj-Desktop-Debug/test_proj exited with code 0
```

# Наследование

- C++ использует 3 основные концепции:
  - Полиморфизм – возможность создавать несколько «вариантов» одной и той же функции
  - Инкапсуляция – мы прячем реализацию внутри класса, сами определяя, что из него можно использовать «снаружи»
  - Наследование




  - Вообще-то есть ещё четвёртая концепция – абстракция

# На примере

Здесь мы используем ещё один элемент стандартной библиотеки – класс std::string

```
#ifndef PERSON_H
#define PERSON_H

#include <string>
using namespace std;

class Person
{
	public:
	Person();
	Person(int age, string name);

	int Age() const {return m_Age;}
	string Name() const {return m_Name;}
	void Greetings() const;

protected:
	int m_Age;
	string m_Name;
};

#endif // PERSON_H
```

# Реализация

```
#include "person.h"
#include <iostream>

Person::Person()
{
	m_Name = "Anonymous";
	m_Age = -1;
}

Person::Person(int age, string name)
{
	m_Age = age;
	m_Name = name;
}

void Person::Greetings() const
{
	cout << "Hello, guys, my name is ”\
 << m_Name << endl;
}
```

# main()

```
#include <iostream>
#include <string>
#include "person.h"

using namespace std;

int main(int argc, char *argv[])
{
      Person anonimous;
      anonimous.Greetings();
      return 0;
}
```

# Добавим класс наследник

Здесь ключевое слово public после имени класса показывает, что используется публичное (открытое) наследование. Класс я добавил в файл person.h

```
class Student : public Person
{
public:
        Student(int age, string name, int mark);
        int Mark() const {return m_AverageMark;}
protected:
        int m_AverageMark;
};

Student::Student(int age, string name, int mark)
        : Person(age, name)
{
        m_AverageMark = mark;
}
```

# Реализация – main.cpp

Здесь для ввода имени используется стандартная объект std::sin;

```
int main(int argc, char *argv[])
{
	Person anonimous;
	anonimous.Greetings();
	string name;
	cout << "Enter student name and press Enter..." << endl;
	cin >> name;
	Student modelStudent (21, name, 5);
	modelStudent.Greetings();
	cout << "Model student " << modelStudent.Name() << \
" has average mark " << modelStudent.Mark() << endl;
	return 0;
}
```

# Ещё один класс

Теперь у моего класса нет новых переменных, но зато я переопределил функцию

```
class Teacher : public Person
{
public:
      Teacher(int age, string name) :
            Person(age, name) {}
      void Greetings();
};


void Teacher::Greetings()
{
      cout << "Hello, students, let's
start our lesson..." << endl;
}
```

# И ещё немного в main()

```
int main(int argc, char *argv[])
{
        Person anonimous;
        anonimous.Greetings();
        string name;
        cout << "Enter student name and press Enter..." << endl;
        cin >> name;
        Student modelStudent (21, name, 5);
        modelStudent.Greetings();
        cout << "Model student " << modelStudent.Name() << \
" has average mark " << modelStudent.Mark() << endl;
        Teacher someone(64, name+"-senior");
        cout << "First, as a teacher: " << endl;
        someone.Greetings();
        cout << "Then, as a person: " << endl;
        someone.Person::Greetings();
        return 0;
}
```

# Что на самом деле создаётся?

```
Person::Person()
{
	m_Name = "Anonymous";
	m_Age = -1;
	cout << "Person constructor called for " << m_Name << endl;
}

Person::Person(int age, string name)
{
	m_Age = age;
	m_Name = name;
	cout << "Person constructor called for " << m_Name << endl;
}
```

# И вернём конструкторы назад

: Person(age, name) – вызов конструктора базового класса

```
Teacher::Teacher(int age, string name)
	: Person(age, name)
{
	cout << "Teacher constructor called for " \
	<< m_Name << endl;
}

Student::Student(int age, string name, int mark)
	: Person(age, name)
{
	m_AverageMark = mark;
	cout << "Student constructor called for " \
	<< m_Name << endl;
}
```

# Всё та же программа...

```
int main(int argc, char *argv[])
{
	Person anonimous;
	anonimous.Greetings();
	string name = "Genady";
	Student modelStudent (21, name, 5);
	modelStudent.Greetings();
	Teacher someone(64, name+"-senior");
	cout << "First, as a teacher: " << endl;
	someone.Greetings();
	cout << "Then, as a person: " << endl;
	someone.Person::Greetings();
	return 0;
}
```

# Смотрим на вывод...

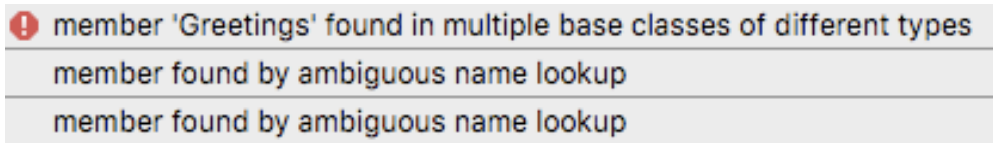
```
Starting /Users/amakashov/projects/build-heir-Desktop-Debug/heir...
Person constructor called for Anonymous
Hello, guys, my name is Anonymous
Person constructor called for Genady
Student constructor called for Genady
Hello, guys, my name is Genady
Person constructor called for Genady-senior
Teacher constructor called for Genady-senior
First, as a teacher:
Hello, students, let's start out lesson...
Then, as a person:
Hello, guys, my name is Genady-senior
Teacher destructor called for Genady-senior
Person destructor called for Genady-senior
Student destructor called for Genady
Person destructor called for Genady
Person destructor called for Anonymous
/Users/amakashov/projects/build-heir-Desktop-Debug/heir exited with code 0
```

# Множественное наследование

```
// в Person.h
class Assistant : public Student, public Teacher
{
public:
	Assistant(int age, string name, int mark);
};

// в Person.cpp
Assistant::Assistant(int age, string name, int mark)
: Student(age, name, mark)
{
	cout << "Assistant constructor called" << endl;
}
```

# Мы получим сообщение об ошибке




- У нас есть функция, определённая в обоих базовых классах
- Компилятор не может понять, какой вариант он должен использовать

# Добавим реализацию

```
// в Person.h
void Greetings()

// в Person.cpp
void Assistant::Greetings()
{
    Student::Greetings();
}
```

# Результат

```
Starting /Users/amakashov/projects/build-heir-Desktop-Debug/heir...
Person constructor called for Anonymous
Student constructor called for Genady-jr.
Person constructor called for Genady-jr.
Teacher constructor called for Genady-jr.
Assistant constructor called
First, as a teacher:
Hello, guys, my name is Genady-jr.
Then, as a person:
Hello, students, let's start out lesson...
Teacher destructor called for Genady-jr.
Person destructor called for Genady-jr.
Student destructor called for Genady-jr.
Person destructor called for Genady-jr.
/Users/amakashov/projects/build-heir-Desktop-Debug/heir exited with code 0
```